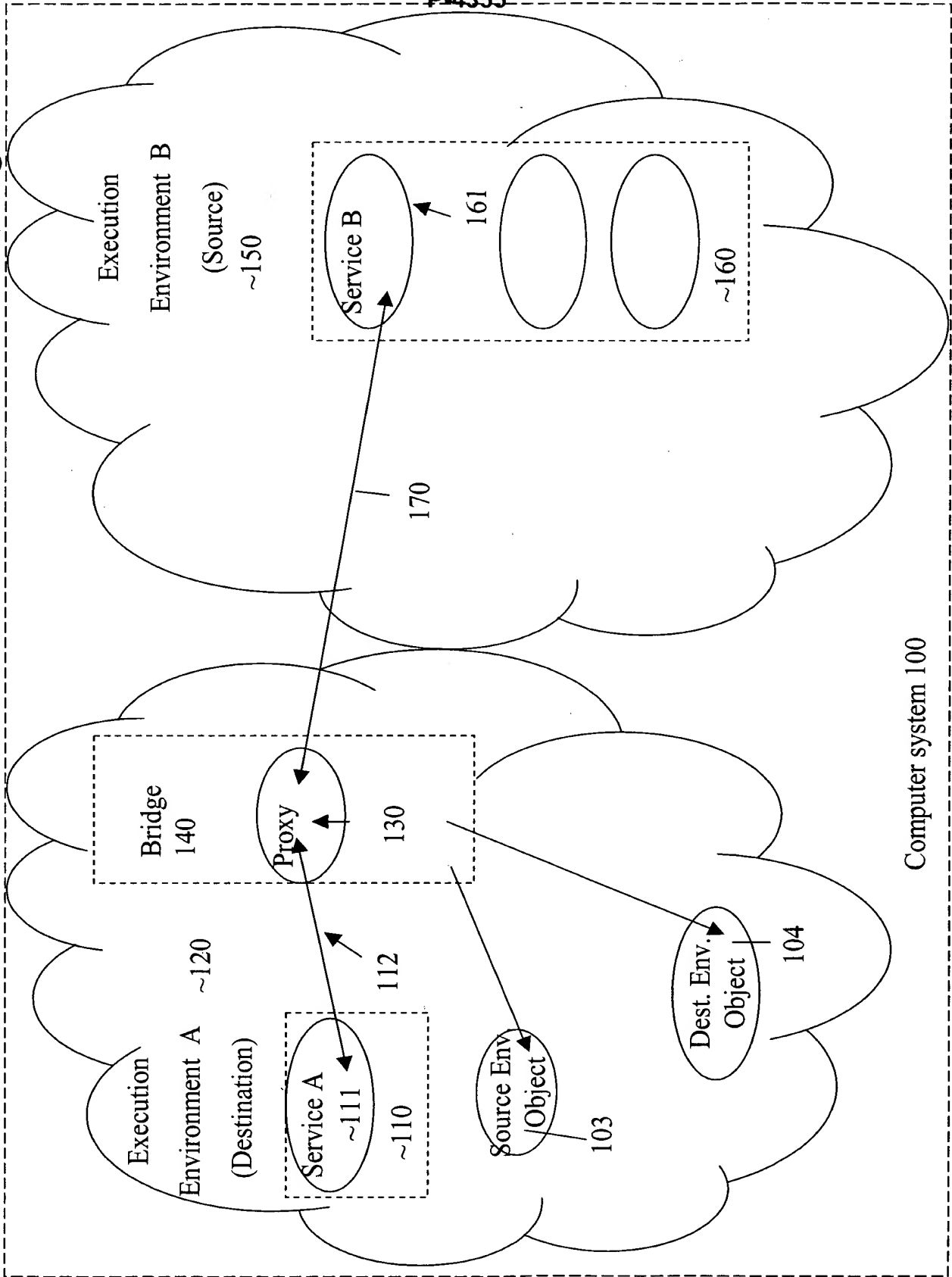
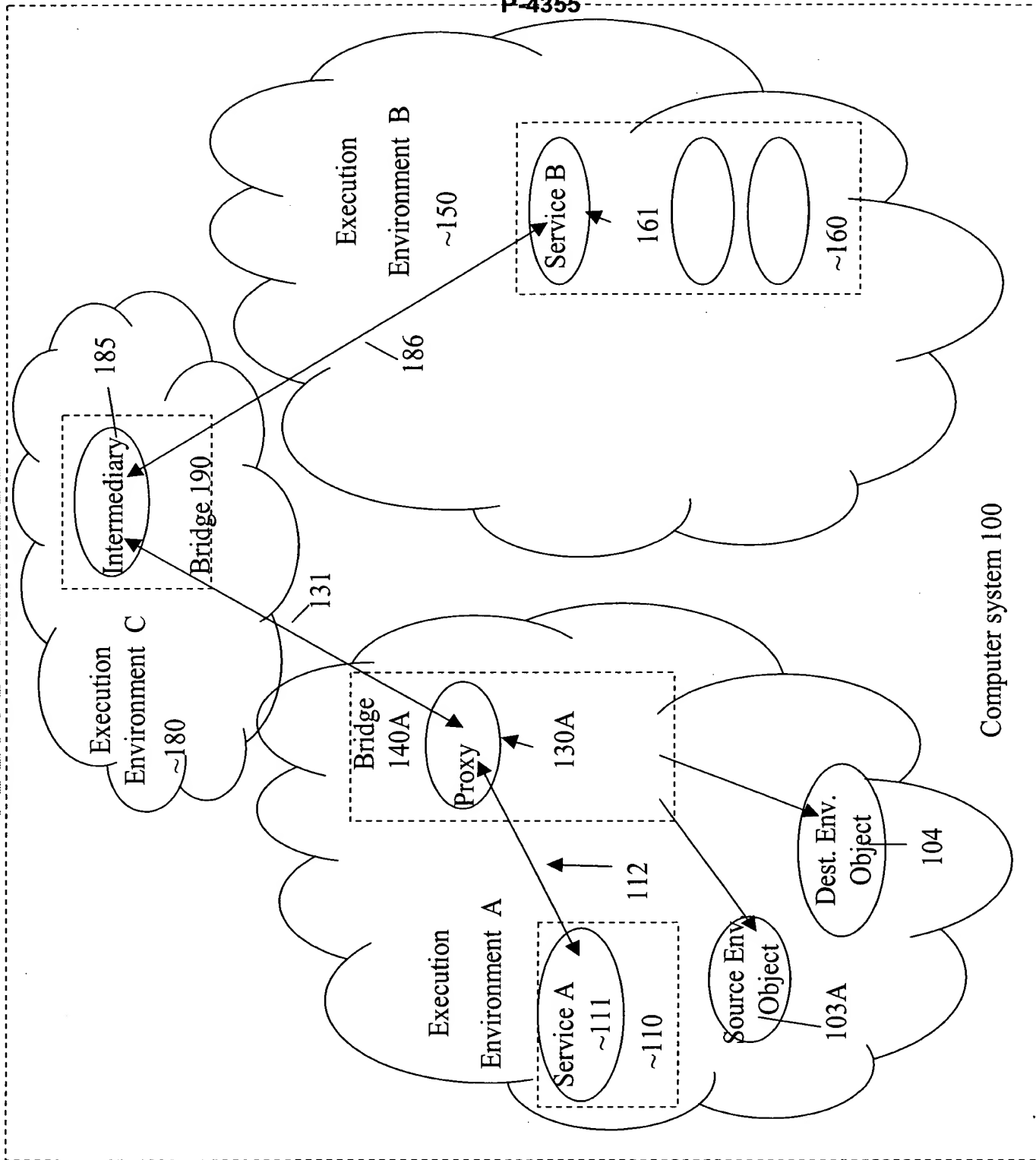


Fig. 1A



T02T10" T2E09/60

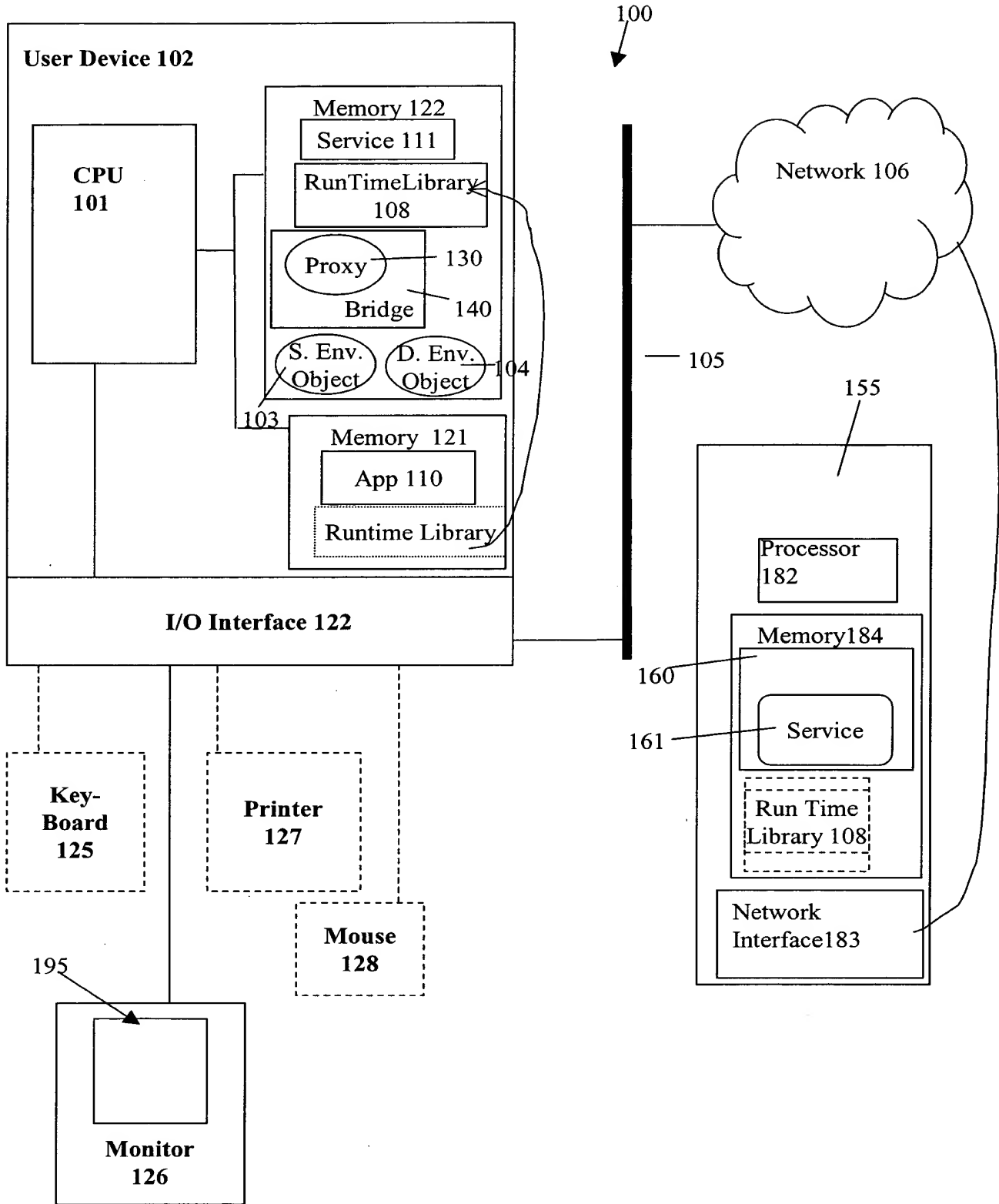
FIG. 1B

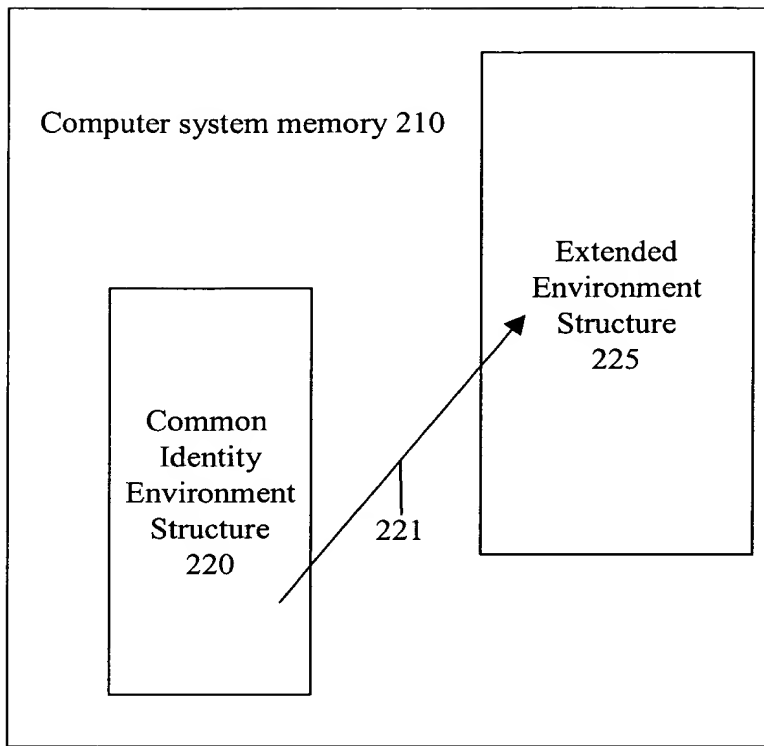


Computer system 100

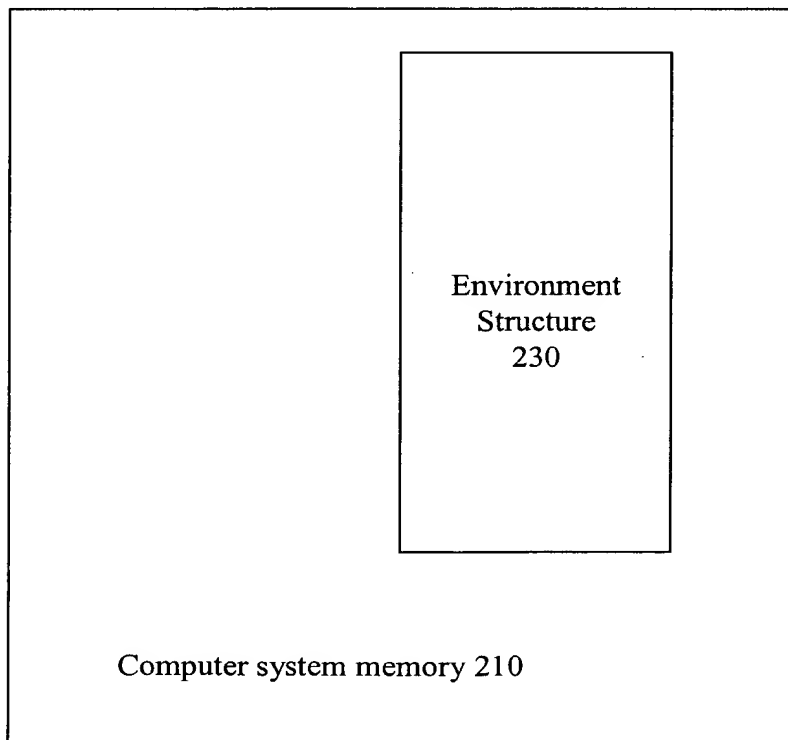
Fig. 1B

Fig. 1C





**FIG. 2A**



**FIG. 2B**

09760321.01201  
T02110 T2E09Z60

5/29

P-4355

```

struct uno_Environment
{
/** a name for this environment
*/
    rtl_String * pName;
/** A free context pointer, that can be used for specific classes of environments, e.g.
    a jvm pointer
*/
    void * pContext;
/** Acquires this environment.
    @param pEnv this interface
*/
    void (SAL_CALL * acquire)( uno_Environment * pEnv );
/** Releases this environment; Last release of environment revokes the environment
    from runtime.
    @param pAccess this access interface
*/
    void (SAL_CALL * release)( uno_Environment * pEnv );
/** Tests if two environments are equal.
    @param pEnv1 one environment
    @param pEnv2 another environment
*/
    sal_Bool (SAL_CALL * equals) ( const uno_Environment* pEnv1, const
        uno_Environment * pEnv2 );
/**
    * You register internal and external interfaces via his method. Internal interfaces are
    proxies that are used in an environment. External interfaces are interfaces
    that are exported to another environment, thus providing an object identifier
    for this task. This can be called an external reference. Interfaces are held
    weakly at an environment; they demand a final revokeInterface() call for each
    interface that has been registered.
        @param pEnv this environment
        @param ppInterface inout parameter for the registered interface
        @param ppOId inout parameter for the corresponding object id
        @param pTypeDescr type description of interface
        @param acquire function to acquire an interface; this function
        provides a boolean return value to signal if the acquisition was
        successful (necessary for proxy interfaces)
*/
    void (SAL_CALL * registerInterface) (uno_Environment * pEnv, void **
        ppInterface, rtl_String ** ppOId, typelib_InterfaceTypeDescription
        *pTypeDescr, uno_regAcquireFunc acquire );
/**
    ANY interface that has been registered is revoked via this method.
        @param pEnv this environment
        @param pOId object id of interface to be revoked
        @param pTypeDescr type description of interface to be revoked
*/
    void (SAL_CALL * revokeInterface) ( uno_Environment * pEnv, rtl_String *
        pOId, typelib_InterfaceTypeDescription * pTypeDescr );

```

Fig. 3A

09760321.011201

6/29

P-4355

09760321 "011201

```
/** Retrieves an interface identified by its object id and type from this environment.
    @param pEnv this environment
    @param ppInterface inout parameter for the registered interface; (0) if
        none was found
    @param pOId object id of interface to be retrieved
    @param pTypeDescr type description of interface to be retrieved
*/
void (SAL_CALL * getRegisteredInterface) ( uno_Environment * pEnv, void
    ** ppInterface, rtl_String * pOId, typelib_InterfaceTypeDescription *
    pTypeDescr );

/**
    Retrieves the object identifier for a registered interface from this environment.
    @param pEnv this environment
    @param ppOId inout parameter for object id of interface; (0) if none
        was found
    @param pInterface a registered interface
    @param pTypeDescr type description of interface
*/
void (SAL_CALL * getRegisteredObjectIdentifier) ( uno_Environment *
    pEnv, rtl_String ** ppOId, void *
    pInterface, typelib_InterfaceTypeDescription * pTypeDescr );

/**
    * Disposing callback function pointer that can be set to get signalled before the
    environment is destroyed.
    @param pEnv environment that is being disposed
*/
void (SAL_CALL * environmentDisposing) ( uno_Environment * pEnv );

/**
    * Computes an object identifier for the given interface; is called by the environment
    implementation.
    @param pEnv corresponding environment
    @param ppOId out param: computed id
    @param pInterface an interface
*/
void (SAL_CALL * computeObjectIdentifier) ( uno_Environment * pEnv,
    rtl_String ** ppOId, void * pInterface );

/** Function to acquire an interface.
    @param pEnv corresponding environment
    @param pInterface an interface
*/
void (SAL_CALL * acquireInterface) ( uno_Environment * pEnv, void *
    pInterface );

/**
    * Function to release an interface.
    @param pEnv corresponding environment
    @param pInterface an interface
*/
void (SAL_CALL * releaseInterface) ( uno_Environment * pEnv, void * pInterface
    );
};
```

Fig. 3B

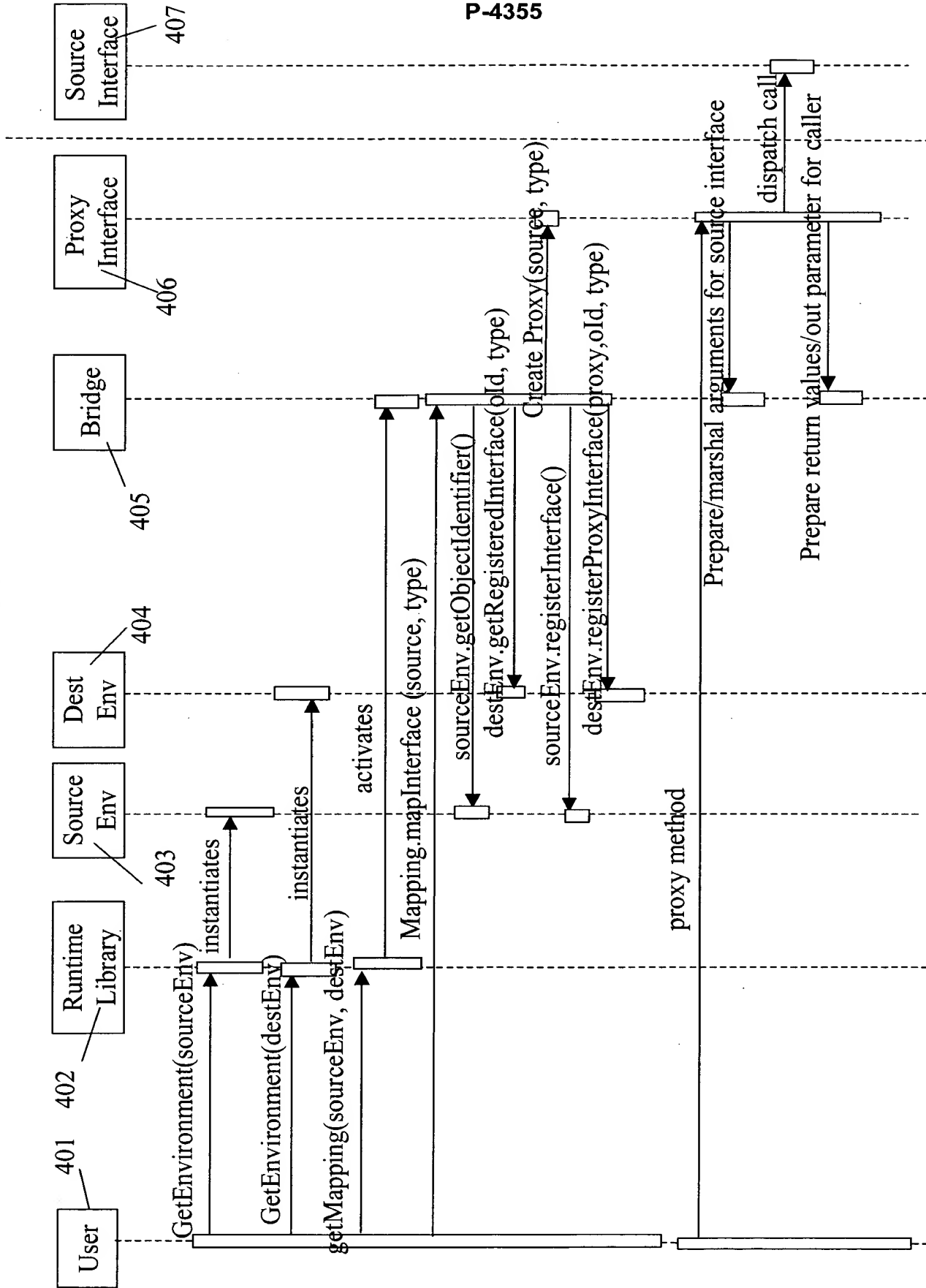


FIG. 4

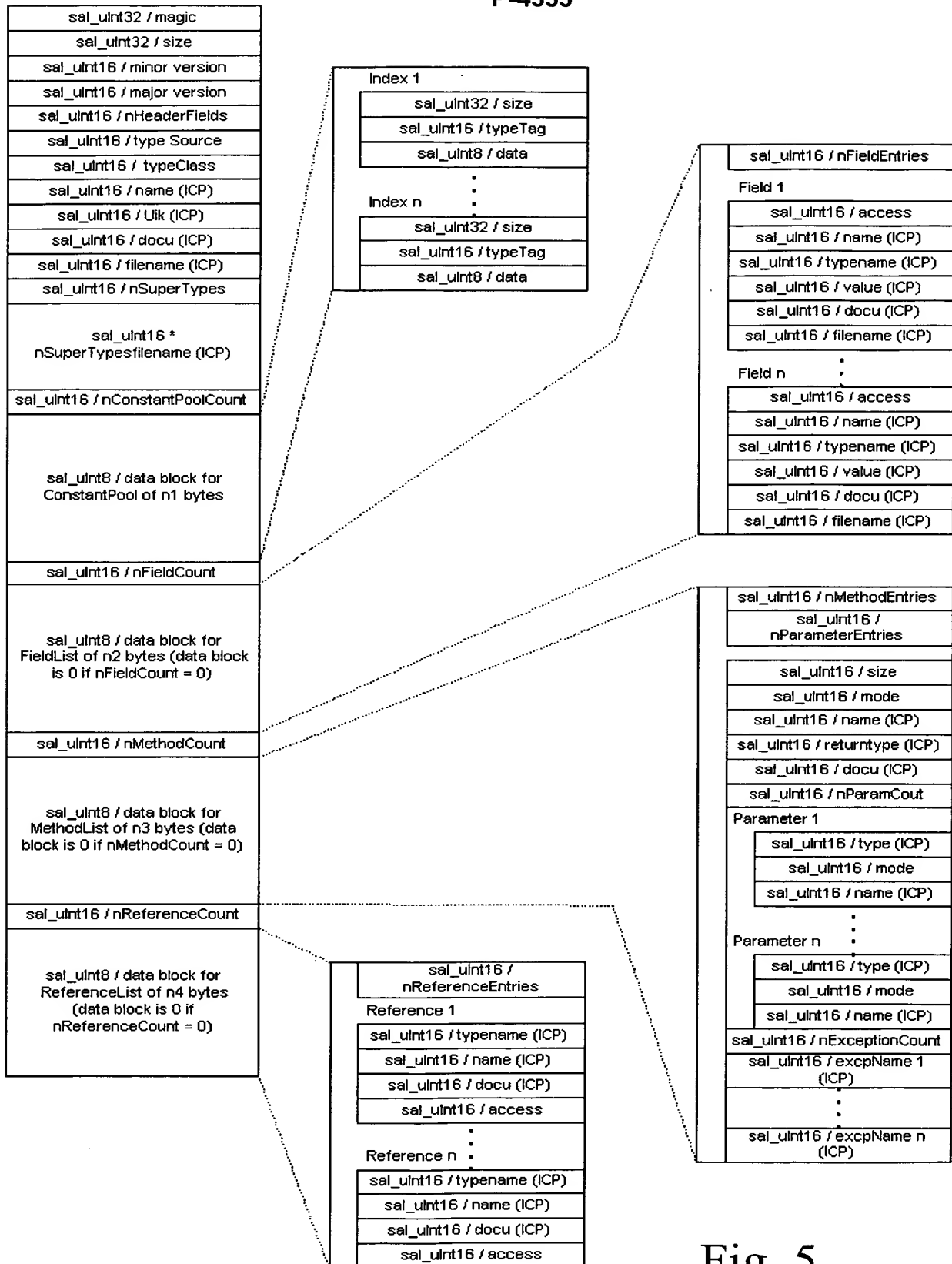


Fig. 5

9/29

P-4355

Offset:	Value
0	Return Address
4	This Pointer
[if struct] 8	[if struct] Pointer to Return Struct
8 12	Parameter 0
. . .	

Memory 610

600

FIG. 6

Offset:	Function pointer:
0	
4	queryInterface() (XInterface member)
8	acquire() (XInterface member)
12	release() (XInterface member)
	bar() (XExample member)
. . .	. . .

Memory 710

700

FIG. 7A

FIG. 6

10/29

P-4355

09760321.011201

Offset:	Function pointer to code:
0	mov eax, 0 jmp cpp_vtable_call
4	mov eax, 1 jmp cpp_vtable_call
8	mov eax, 2 jmp cpp_vtable_call
...	...

Memory 730

720

FIG. 7B

11/29

P-4355

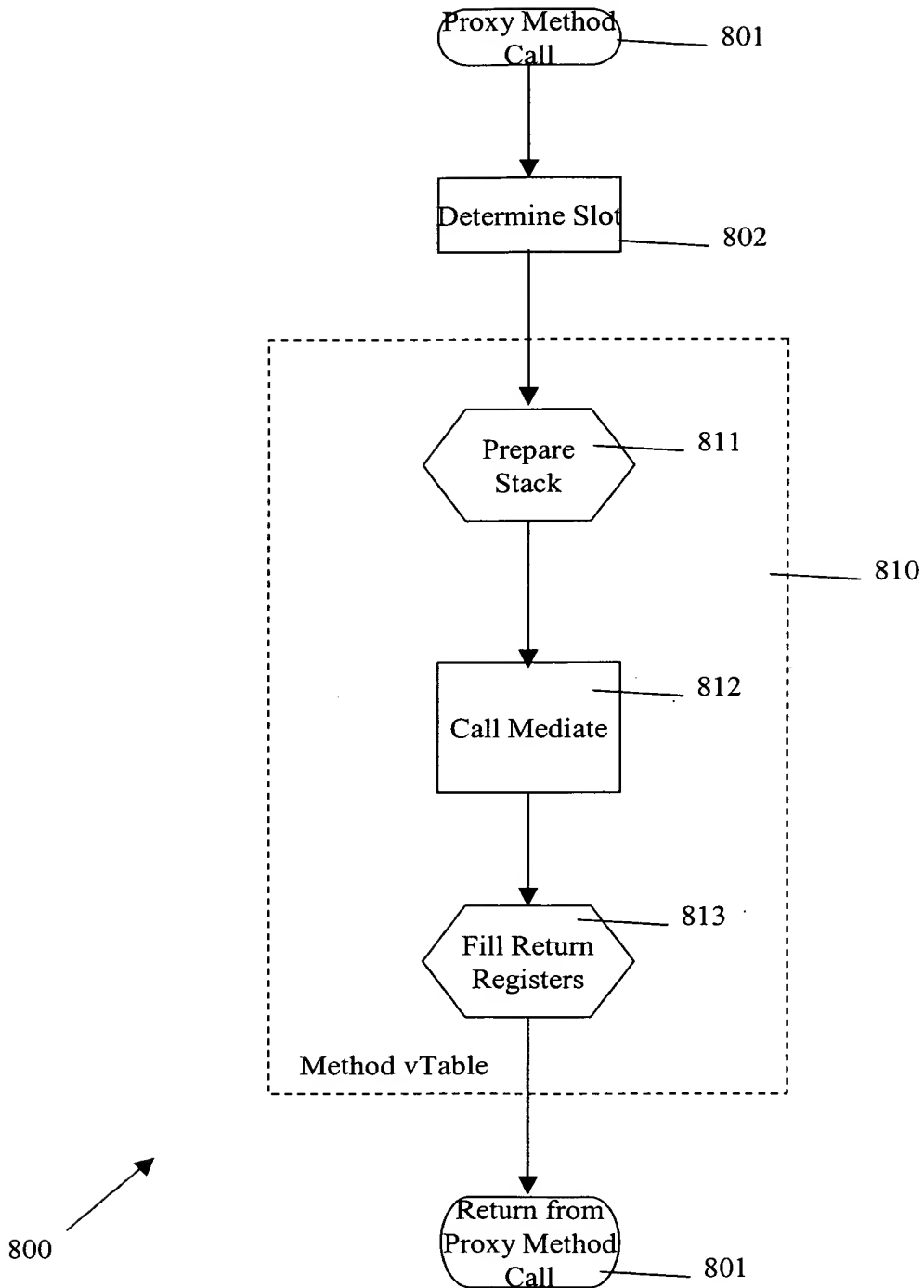


FIG. 8

12/29

P-4355

FIG. 9

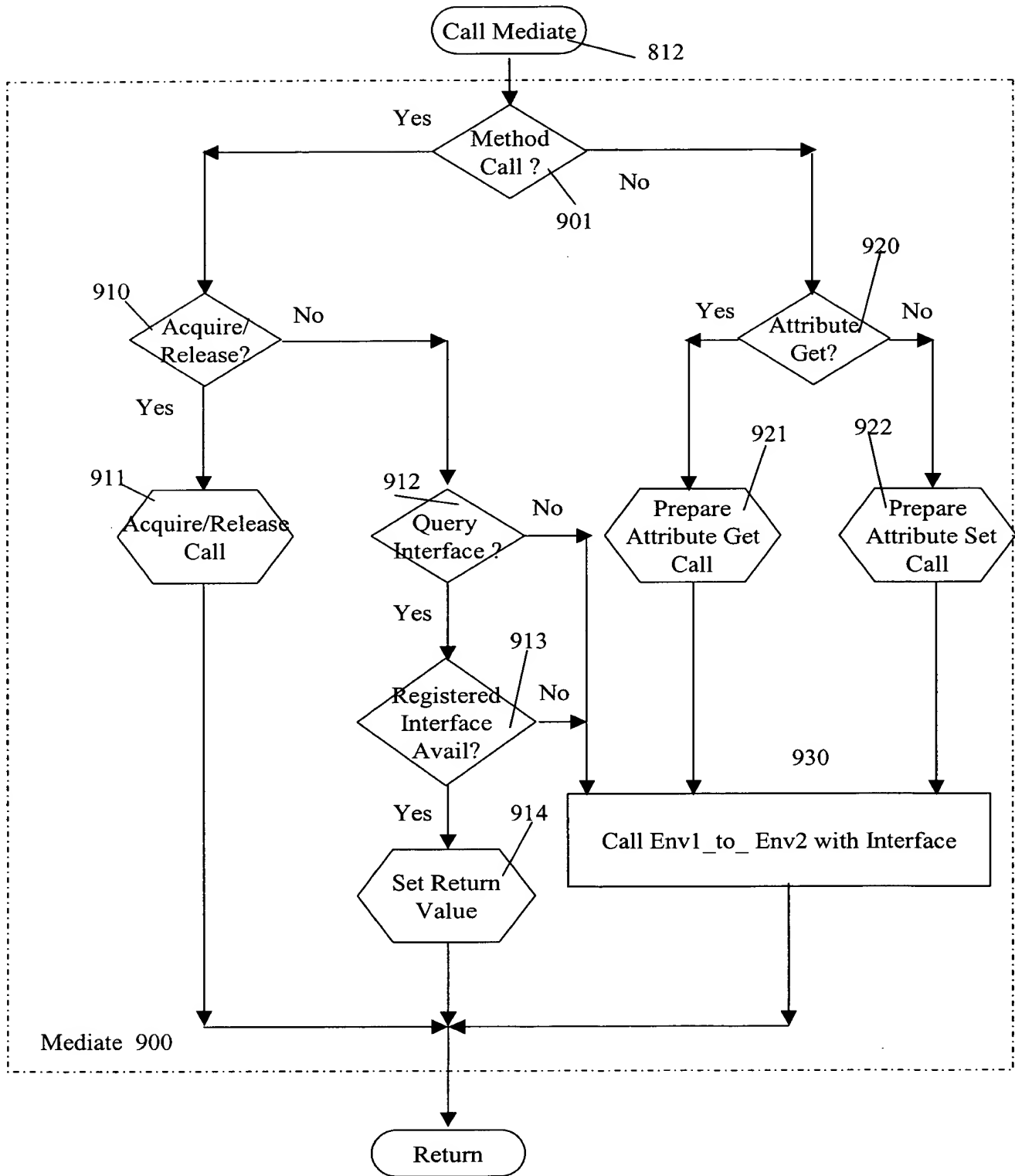


FIG. 9

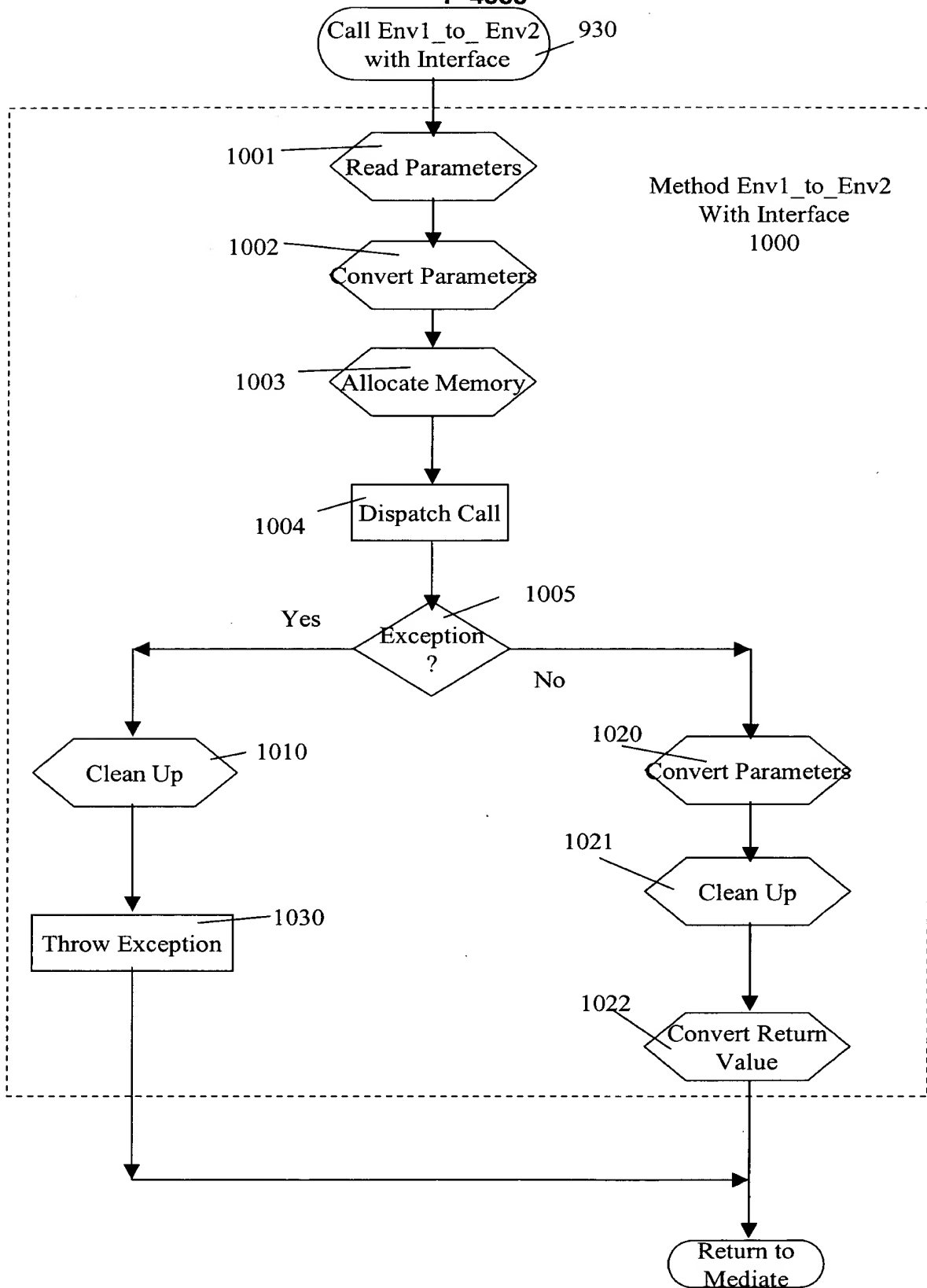


FIG. 10

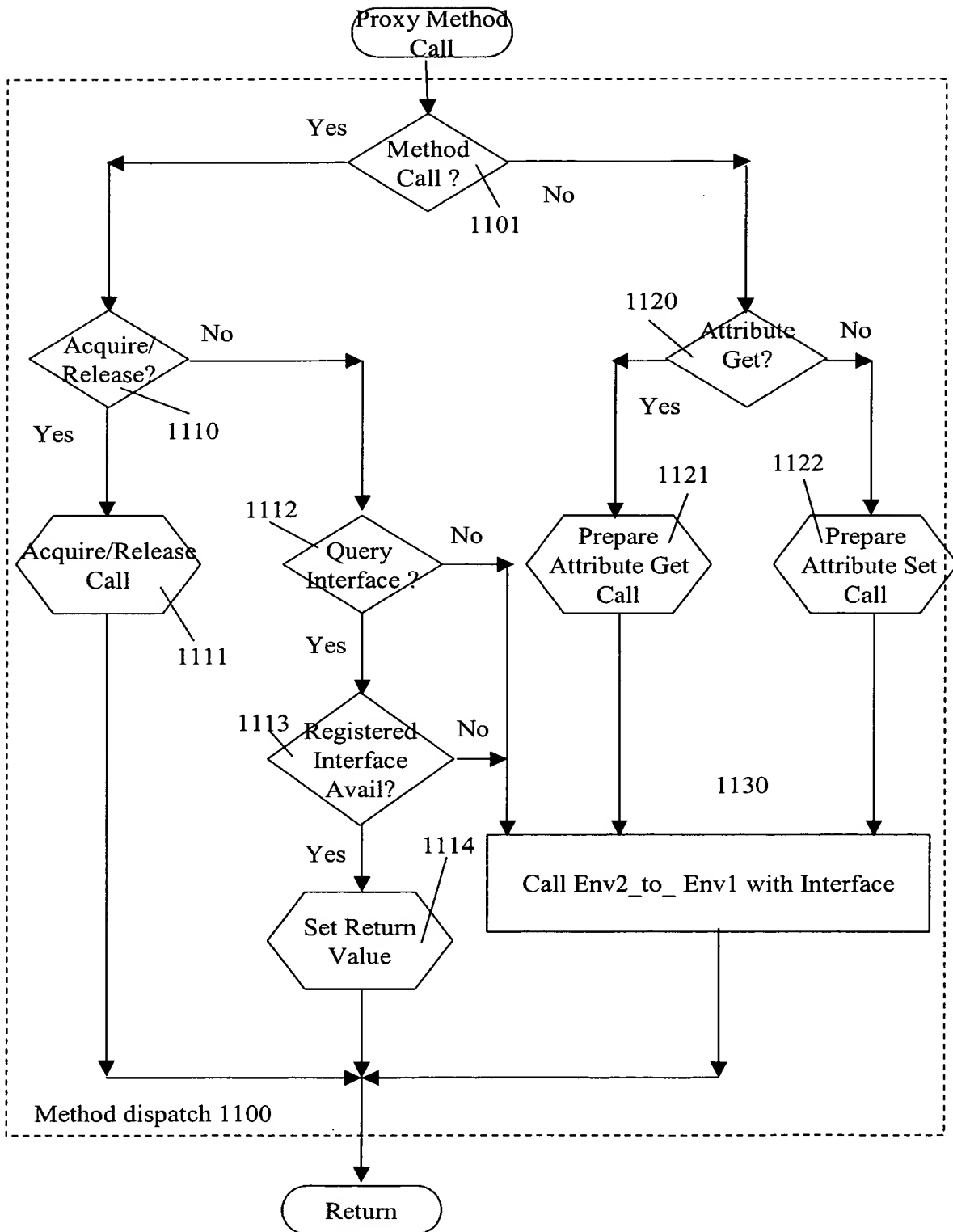


FIG. 11

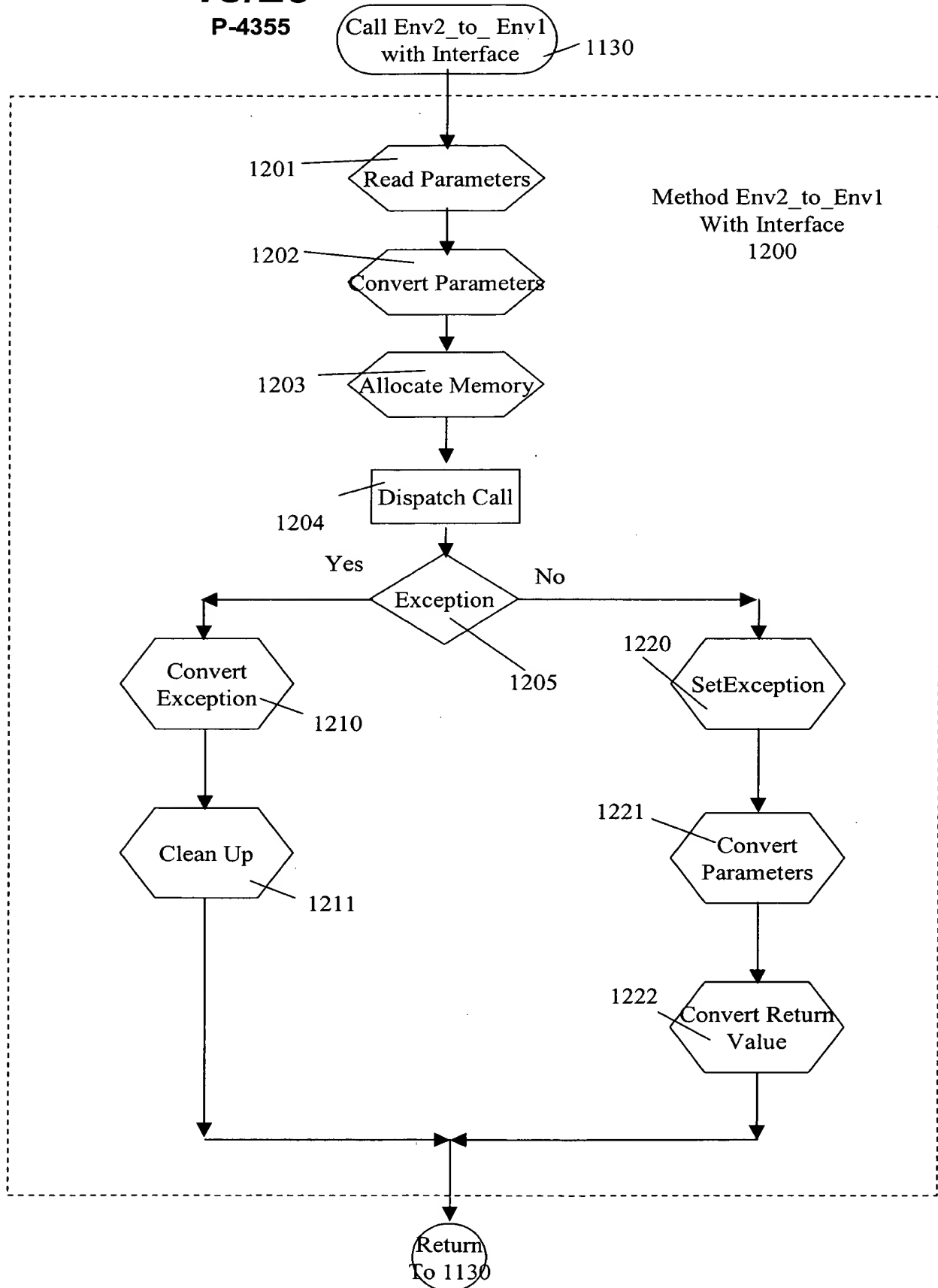


FIG. 12

FIG. 13A

```

inline void SAL_CALL cppu_Mapping_uno2cpp(
    uno_Mapping * pMapping, void ** ppCppI,
    void * pUnoI, typelib_InterfaceTypeDescription * pTypeDescr )
{
    OSL_ASSERT( ppCppI && pTypeDescr );
    if ( *ppCppI )
    {
        reinterpret_cast< ::com::sun::star::uno::XInterface * >( *ppCppI )->release();
        *ppCppI = 0;
    }
    if ( pUnoI )
    {
        cppu_Bridge * pBridge = static_cast< cppu_Mapping * >( pMapping )->pBridge;

        // get object id of uno interface to be wrapped
        rtl_uString * pOid = 0;
        (*pBridge->pUnoEnv->getObjectIdentifier)( pBridge->pUnoEnv, &pOid, pUnoI );
        OSL_ASSERT( pOid );

        // try to get any known interface from target environment
        (*pBridge->pCppEnv->getRegisteredInterface)(
            pBridge->pCppEnv, ppCppI, pOid, pTypeDescr );

        if (! *ppCppI) // no existing interface, register new proxy interface
        {
            // try to publish a new proxy (ref count initially 1)
            cppu_cppInterfaceProxy * pProxy = new cppu_cppInterfaceProxy(
                pBridge, reinterpret_cast< uno_Interface * >( pUnoI ), pTypeDescr, pOid );
            ::com::sun::star::uno::XInterface * pSurrogate = pProxy;
            cppu_cppInterfaceProxy_patchVtable( pSurrogate, pProxy->pTypeDescr );
        }
    }
}

```

FIG. 13B

```

// proxy may be exchanged during registration
(*pBridge->pCppEnv->registerProxyInterface)(
    pBridge->pCppEnv, reinterpret_cast<void **>( &pSurrogate ),
    cppu_cppInterfaceProxy_free, pOid, pTypeDescr );

    *ppCppI = pSurrogate;
}
::rtl_ustring_release( pOid );
}
}

```

FIG. 14

```

//inline void SAL_CALL cppu_cppInterfaceProxy_free( uno_ExtEnvironment * pEnv, void * pProxy )
{
    cppu_cppInterfaceProxy * pThis =
        static_cast< cppu_cppInterfaceProxy * >(
            reinterpret_cast< ::com::sun::star::uno::XInterface * >( pProxy ) );
    OSL_ASSERT( pEnv == pThis->pBridge->pCppEnv );

    (*pThis->pBridge->pUnoEnv->revokeInterface)( pThis->pBridge->pUnoEnv, pThis->pUnoI );
    (*pThis->pUnoI->release)( pThis->pUnoI );
    ::typelib_typedescription_release( (typelib_TypeDescription *)pThis->pTypeDescr );
    pThis->pBridge->release();

#ifdef DEBUG
    *(int *)pProxy = 0xdeadbabe;
#endif
    delete pThis;
}

```

FIG. 15

```

inline void cppu_cppInterfaceProxy::acquireProxy()
{
    if (1 == osl_incrementInterlockedCount( &nRef ))
    {
        // rebirth of proxy zombie
        // register at cpp env
        void * pThis = static_cast< ::com::sun::star::uno::XInterface * >( this );
        (*pBridge->pCppEnv->registerProxyInterface)(
            pBridge->pCppEnv, &pThis, cppu_cppInterfaceProxy_free, oid.pData, pTypeDescr );
        OSL_ASSERT( pThis == static_cast< ::com::sun::star::uno::XInterface * >( this ) );
    }
}

inline void cppu_cppInterfaceProxy::releaseProxy()
{
    if (! osl_decrementInterlockedCount( &nRef )) // last release
    {
        // revoke from cpp env
        (*pBridge->pCppEnv->revokeInterface)(
            pBridge->pCppEnv, static_cast< ::com::sun::star::uno::XInterface * >( this ) );
    }
}

inline cppu_cppInterfaceProxy::cppu_cppInterfaceProxy(
    cppu_Bridge * pBridge_, uno_Interface * pUnoI_,
    typelib_InterfaceTypeDescription * pTypeDescr_, const ::rtl::OUString & rOid_
    : nRef( 1 )
    , pBridge( pBridge_ )
    , pUnoI( pUnoI_ )
    , pTypeDescr( pTypeDescr_ )
    , oid( rOid_ )
)
{
    pBridge->acquire();
    ::typelib_typedescription_acquire( (typelib_TypeDescription *)pTypeDescr );
    if (! ((typelib_TypeDescription *)pTypeDescr)->bComplete)
        ::typelib_typedescription_complete( (typelib_TypeDescription **)&pTypeDescr );
    (*pBridge->pUnoEnv->registerInterface)(
        pBridge->pUnoEnv, reinterpret_cast< void ** >( &pUnoI ), oid.pData, pTypeDescr );
    (*pUnoI->acquire)( pUnoI );
}

```

FIG. 16A

```

inline void SAL_CALL cppu_unoInterfaceProxy_free( uno_ExtEnvironment * pEnv, void * pProxy )
{
    cppu_unoInterfaceProxy * pThis =
        static_cast< cppu_unoInterfaceProxy * >(
            reinterpret_cast< uno_Interface * >( pProxy ) );
    OSL_ASSERT( pEnv == pThis->pBridge->pUnoEnv );

    (*pThis->pBridge->pCppEnv->revokeInterface)( pThis->pBridge->pCppEnv, pThis->pCppl );
    pThis->pCppl->release();
    ::typelib_typedescription_release( (typelib_TypeDescription *)pThis->pTypeDescr );
    pThis->pBridge->release();

#ifdef DEBUG
    *(int *)pProxy = 0xdeadbeef;
#endif
    delete pThis;
}
inline void SAL_CALL cppu_unoInterfaceProxy_acquire( uno_Interface * pUnoI )
{
    if ( 1 == osl_incrementInterlockedCount( & static_cast< cppu_unoInterfaceProxy * >( pUnoI )->nRef ) )
    {
        // rebirth of proxy zombie
        // register at uno env
        void * pThis = pUnoI;
    }
#ifdef DEBUG
    #endif
    (*static_cast< cppu_unoInterfaceProxy * >( pUnoI )->pBridge->pUnoEnv->registerProxyInterface)(
        static_cast< cppu_unoInterfaceProxy * >( pUnoI )->pBridge->pUnoEnv,
        reinterpret_cast< void ** >( &pUnoI ), cppu_unoInterfaceProxy_free,
        static_cast< cppu_unoInterfaceProxy * >( pUnoI )->oid.pData,
        static_cast< cppu_unoInterfaceProxy * >( pUnoI )->pTypeDescr );
}

```

FOUO" 22E09/60

FIG. 16B

```

#ifdef DEBUG
    OSL_ASSERT( pThis == pUnoI );
#endif
}
inline void SAL_CALL cppu_unoInterfaceProxy_release( uno_Interface * pUnoI )
{
    if ( ! osl_decrementInterlockedCount( & static_cast< cppu_unoInterfaceProxy * >( pUnoI )->nRef ) )
    {
        // revoke from uno env on last release
        (*static_cast< cppu_unoInterfaceProxy * >( pUnoI )->pBridge->pUnoEnv->revokeInterface)(
            static_cast< cppu_unoInterfaceProxy * >( pUnoI )->pBridge->pUnoEnv, pUnoI );
    }
}

```

FIG. 17A

```

inline void SAL_CALL cppu_Mapping_cpp2uno(
    uno_Mapping * pMapping, void ** ppUnoI,
    void * pCppl, typelib_InterfaceTypeDescription * pTypeDescr )
{
    OSL_ENSURE( ppUnoI && pTypeDescr, "### null ptr!" );
    if ( *ppUnoI )
    {
        (*reinterpret_cast< uno_Interface * >( *ppUnoI )->release)(
            reinterpret_cast< uno_Interface * >( *ppUnoI ) );
        *ppUnoI = 0;
    }
    if ( pCppl )

```

FIG. 17B

```

{
    cppu_Bridge * pBridge = static_cast< cppu_Mapping * >( pMapping )->pBridge;

    // get object id of interface to be wrapped
    rtl_uString * pOID = 0;
    (*pBridge->pCppEnv->getObjectIdentifier)( pBridge->pCppEnv, &pOID, pCppI );
    OSL_ASSERT( pOID );

    // try to get any known interface from target environment
    (*pBridge->pUnoEnv->getRegisteredInterface)(
        pBridge->pUnoEnv, ppUnOI, pOID, pTypeDescr );

    if (! *ppUnOI) // no existing interface, register new proxy interface
    {
        // try to publish a new proxy (refcount initially 1)
        uno_Interface * pSurrogate = new cppu_unoInterfaceProxy(
            pBridge, reinterpret_cast< :com::sun::star::uno::XInterface * >( pCppI ),
            pTypeDescr, pOID );

        // proxy may be exchanged during registration
        (*pBridge->pUnoEnv->registerProxyInterface)(
            pBridge->pUnoEnv, reinterpret_cast< void ** >( &pSurrogate ),
            cppu_unoInterfaceProxy_free, pOID, pTypeDescr );

        *ppUnOI = pSurrogate;
    }
    ::rtl_ustring_release( pOID );
}
}

```

FIG. 18

```

inline cppu_unoInterfaceProxy::cppu_unoInterfaceProxy(
    cppu_Bridge * pBridge_, ::com::sun::star::uno::XInterface * pCppl_,
    typelib_InterfaceTypeDescription * pTypeDescr_, const :rtl::OUString & rOid_)
: nRef( 1 )
, pBridge( pBridge_ )
, pCppl( pCppl_ )
, pTypeDescr( pTypeDescr_ )
, oid( rOid_ )
{
    pBridge->acquire();
    ::typelib_typedescription_acquire( (typelib_TypeDescription *)pTypeDescr );
    if ( ! ((typelib_TypeDescription *)pTypeDescr)->bComplete )
        ::typelib_typedescription_complete( (typelib_TypeDescription **)&pTypeDescr );
    (*pBridge->pCpplEnv->registerInterface)(
        pBridge->pCpplEnv, reinterpret_cast< void ** >( &pCppl ), oid.pData, pTypeDescr );
    pCppl->acquire();

    // uno_Interface
    uno_Interface::acquire = CPPU_CURRENT_NAMESPACE::cppu_unoInterfaceProxy_acquire;
    uno_Interface::release = CPPU_CURRENT_NAMESPACE::cppu_unoInterfaceProxy_release;
    uno_Interface::pDispatcher = CPPU_CURRENT_NAMESPACE::cppu_unoInterfaceProxy_dispatch;
}
//-----
inline void SAL_CALL cppu_Mapping_acquire( uno_Mapping * pMapping )
{
    static_cast< cppu_Mapping * >( pMapping )->pBridge->acquire();
}
//-----
inline void SAL_CALL cppu_Mapping_release( uno_Mapping * pMapping )
{
    static_cast< cppu_Mapping * >( pMapping )->pBridge->release();
}

```

FOUO "T2E09/60

FIG. 19

```

//
inline cppu_Mapping::cppu_Mapping( cppu_Bridge * pBridge_, uno_MapInterfaceFunc fpMap )
: pBridge( pBridge_ )
{
    uno_Mapping::acquire = cppu_Mapping_acquire;
    uno_Mapping::release = cppu_Mapping_release;
    uno_Mapping::mapInterface = fpMap;
}
//
inline cppu_Bridge::cppu_Bridge( uno_ExtEnvironment * pCppEnv_, uno_ExtEnvironment * pUnoEnv_,
                                sal_Bool bExportCpp2Uno_ )
: nRef( 1 )
, pCppEnv( pCppEnv_ )
, pUnoEnv( pUnoEnv_ )
, aCpp2Uno( this, cppu_Mapping_cpp2uno )
, aUno2Cpp( this, cppu_Mapping_uno2cpp )
, bExportCpp2Uno( bExportCpp2Uno_ )
{
    *((uno_Environment *)pCppEnv)->acquire)( (uno_Environment *)pCppEnv );
    *((uno_Environment *)pUnoEnv)->acquire)( (uno_Environment *)pUnoEnv );
}
//
inline void SAL_CALL cppu_Bridge_free( uno_Mapping * pMapping )
{
    cppu_Bridge * pThis = static_cast< cppu_Mapping * >( pMapping )->pBridge;
    *((uno_Environment *)pThis->pUnoEnv)->release)( (uno_Environment *)pThis->pUnoEnv );
    *((uno_Environment *)pThis->pCppEnv)->release)( (uno_Environment *)pThis->pCppEnv );
    delete pThis;
}

```

FIG. 20

```

inline void cppu_Bridge::acquire()
{
    if (1 == osl_incrementInterlockedCount( &nRef ))
    {
        if (bExportCpp2Uno)
        {
            uno_Mapping * pMapping = &aCpp2Uno;
            uno_registerMapping( &pMapping, cppu_Bridge_free,
                                (uno_Environment *)pCppEnv, (uno_Environment *)pUnoEnv, 0 );
        }
        else
        {
            uno_Mapping * pMapping = &aUno2Cpp;
            uno_registerMapping( &pMapping, cppu_Bridge_free,
                                (uno_Environment *)pUnoEnv, (uno_Environment *)pCppEnv, 0 );
        }
    }
}

//
inline void cppu_Bridge::release()
{
    if (! osl_decrementInterlockedCount( &nRef ))
    {
        uno_revokeMapping( bExportCpp2Uno ? &aCpp2Uno : &aUno2Cpp );
    }
}

```

FIG. 21

```

inline void SAL_CALL cppu_ext_getMapping(
{
    uno_Mapping ** ppMapping, uno_Environment * pFrom, uno_Environment * pTo )
{
    OSL_ASSERT( ppMapping && pFrom && pTo );
    if (ppMapping && pFrom && pTo && pFrom->pExtEnv && pTo->pExtEnv)
    {
        uno_Mapping * pMapping = 0;

        if (0 == rtl_ustr_ascii_compare( pFrom->pTypeName->buffer,
CPPU_CURRENT_LANGUAGE_BINDING_NAME ) &&
            0 == rtl_ustr_ascii_compare( pTo->pTypeName->buffer, UNO_LB_UNO ))
        {
            // ref count initially 1
            pMapping = &(new cppu_Bridge( pFrom->pExtEnv, pTo->pExtEnv, sal_True ))-
>aCpp2Uno;

            ::uno_registerMapping( &pMapping, cppu_Bridge_free,
                (uno_Environment *)pFrom->pExtEnv,
                (uno_Environment *)pTo->pExtEnv, 0 );
        }
        if (0 == rtl_ustr_ascii_compare( pTo->pTypeName->buffer,
CPPU_CURRENT_LANGUAGE_BINDING_NAME ) &&
            0 == rtl_ustr_ascii_compare( pFrom->pTypeName->buffer, UNO_LB_UNO ))
        {
            // ref count initially 1
            pMapping = &(new cppu_Bridge( pTo->pExtEnv, pFrom->pExtEnv, sal_False ))-
>aUno2Cpp;

            ::uno_registerMapping( &pMapping, cppu_Bridge_free,
                (uno_Environment *)pFrom->pExtEnv,
                (uno_Environment *)pTo->pExtEnv, 0 );
        }
    }

    if (*ppMapping)
        (*(ppMapping)->release)( *ppMapping );
    *ppMapping = pMapping;
}
}

```

FIG. 22A

```

#if (defined(__SUNPRO_CC) && (__SUNPRO_CC == 0x500)) || (defined(__GNUC__) &&
    defined(__APPLE__))
    static ::rtl::OUString * s_pStaticOidPart = 0;
#endif

// environment init stuff
//-----
----
inline const ::rtl::OUString & SAL_CALL cppenv_getStaticOidPart()
{
    #if ! ((defined(__SUNPRO_CC) && (__SUNPRO_CC == 0x500)) || (defined(__GNUC__) &&
        defined(__APPLE__)))
        static ::rtl::OUString * s_pStaticOidPart = 0;
    #endif
    if (! s_pStaticOidPart)
    {
        ::osl::MutexGuard aGuard( ::osl::Mutex::getGlobalMutex() );
        if (! s_pStaticOidPart)
        {
            ::rtl::OUStringBuffer aRet( 64 );
            aRet.appendAscii( RTL_CONSTASCII_STRINGPARAM("]") );
            // pid
            oslProcessInfo info;
            info.Size = sizeof(oslProcessInfo);
            if (::osl_getProcessInfo( 0, osl_Process_IDENTIFIER, &info ) ==
                osl_Process_E_None)
            {
                aRet.append( (sal_Int64)info.Ident, 16 );
            }
            else
            {
                aRet.appendAscii( RTL_CONSTASCII_STRINGPARAM("unknown process id") );
            }
        }
    }
}

```

FIG. 22B

```

// good guid
sal_uint8 ar[16];
::rtl_getGlobalProcessId( ar );
aRet.append( (sal_Unicode)'' );
for ( sal_Int32 i = 0; i < 16; ++i )
{
    aRet.append( (sal_Int32)ar[i], 16 );
}
#if (defined( __SUNPRO_CC ) && ( __SUNPRO_CC == 0x500 ) || (defined( __GNUC__ ) &&
defined( __APPLE__ ) )
    s_pStaticOidPart = new ::rtl::OUString( aRet.makeStringAndClear() );
#else
    static ::rtl::OUString s_aStaticOidPart( aRet.makeStringAndClear() );
    s_pStaticOidPart = &s_aStaticOidPart;
#endif
}
return *s_pStaticOidPart;
}

```

FIG. 23

```

// functions set at environment init
//-----
inline void SAL_CALL cppu_cppenv_computeObjectIdentifier(
    uno_ExtEnvironment * pEnv, rtl_uString ** ppOid, void * pInterface )
{
    OSL_ENSURE( pEnv && ppOid && pInterface, "### null ptr!" );
    if (pEnv && ppOid && pInterface)
    {
        if (*ppOid)
        {
            rtl_uString_release( *ppOid );
            *ppOid = 0;
        }
        ::com::sun::star::uno::Reference< ::com::sun::star::uno::XInterface > xHome(
            reinterpret_cast< ::com::sun::star::uno::XInterface * >( pInterface ),
            ::com::sun::star::uno::UNO_QUERY );
        OSL_ENSURE( xHome.is(), "### query to XInterface failed!" );
        if (xHome.is())
        {
            // interface
            ::rtl::OUStringBuffer oid( 64 );
            oid.append( (sal_Int64)xHome.get(), 16 );
            oid.append( (sal_Unicode)';' );
            // environment[context]
            oid.append( (uno_Environment *)pEnv->pTypeName );
            oid.append( (sal_Unicode)'\[' );
            oid.append( (sal_Int64)(uno_Environment *)pEnv->pContext, 16 );
            // process;good guid
            oid.append( cppu_cppenv_getStaticOidPart() );
            ::rtl::OUString aRet( oid.makeStringAndClear() );
            ::rtl_uString_acquire( *ppOid = aRet.pData );
        }
    }
}

```

FIG. 24

```

inline void SAL_CALL cppu_cppenv_acquireInterface( uno_ExtEnvironment *, void * pCppl )
{
    reinterpret_cast< ::com::sun::star::uno::XInterface * >( pCppl )->acquire();
}
//-----
inline void SAL_CALL cppu_cppenv_releaseInterface( uno_ExtEnvironment *, void * pCppl )
{
    reinterpret_cast< ::com::sun::star::uno::XInterface * >( pCppl )->release();
}
//-----
inline void SAL_CALL cppu_cppenv_initEnvironment( uno_Environment * pCpplEnv )
{
    OSL_ENSURE( pCpplEnv->pExtEnv, "### expected extended environment!" );
    OSL_ENSURE( rtl_ustring_ascii_compare( pCpplEnv->pTypeName->buffer, CPPU_CURRENT_LANGUAGE_BINDING_NAME ) ==
0,
        "### wrong environment type!" );
    ((uno_ExtEnvironment *)pCpplEnv)->computeObjectIdentifier =
CPPU_CURRENT_NAMESPACE::cppu_cppenv_computeObjectIdentifier;
    ((uno_ExtEnvironment *)pCpplEnv)->acquireInterface =
CPPU_CURRENT_NAMESPACE::cppu_cppenv_acquireInterface;
    ((uno_ExtEnvironment *)pCpplEnv)->releaseInterface =
CPPU_CURRENT_NAMESPACE::cppu_cppenv_releaseInterface;
}
}
#endif

```